SQL Injection security flaw in OpenEMR medical records system.

12 June 2014 · security, vulnerability

I recently examined a popular open source medical records system named OpenEMR (http://www.open-emr.org/). A quick review of the app uncovered a SQL Injection (https://www.owasp.org/index.php/SQL_Injection) vulnerability in the application, that would allow an attacker to execute their own SQL commands against the system. The attack is relatively textbook and its detection and exploitation are outlined below. Firstly, a description of the product:

> **Profile: OpenEMR** *is a medical practice management software which also supports Electronic Medical Records (EMR). It is ONC Complete Ambulatory EHR certified and it features fully integrated electronic medical records, practice management for a medical practice, scheduling and electronic billing.
>
> The server side is written in PHP and can be employed in conjunction with a LAMP “stack”, though any operating systems with PHP-support are also supported.
> …
> In the US, it has been estimated that there are more than 5,000 installations of OpenEMR in physician offices and other small healthcare facilities serving more than 30 million patients. Internationally, it has been estimated that OpenEMR is installed in over 15,000 healthcare facilities, translating into more than 45,000 practitioners using the system which are serving greater than 90 million patients.*
>
> Source: Wikipedia: http://en.wikipedia.org/wiki/OpenEMR (http://en.wikipedia.org/wiki/OpenEMR)

Affected versions: OpenEMR 4.1.2 Patch 5 (and likely previous patches & releases)

Fix in: OpenEMR 4.1.2 Patch 6

As usual I reviewed the system as a user, browsing features and recording my actions in my intercepting proxy (BurpSuite). This gave me a good idea of the default system features and usage model. Combined with review through the online documentation, I gained a broad idea of how the system is used and its features or ‘claims’.

The latest/patched code was relatively well protected against SQL Injection, with widespread use of prepared statements, a good defence against 1st order SQL Injection. But, I noticed a few queries were not parameterised. While this is not necessarily a problem, if its possible to include custom inputs into the query, then vulnerabilities can creep in.

In this case, the affected query was a delete for ‘Patient Disclosures’. When the user opts to delete a Disclosure record via the user interface the system runs this query, inserting the record identifier sent via the browser.

Unfortunately, the Open EMR system does not filter out inappropriate characters for these requests, meaning SQL can be written unmodified into the request. As long as the SQL, when combined with the remainder of the query is valid syntactically, the query is then executed. If code had restricted the input to be, for example positive integers, then this vulnerability would be largely mitigated.

You can see the vulnerable code here:

File: openemr-4.1.2/library/log.inc

```
function deleteDisclosure($deletelid)

{

        $sql="delete from extended_log where id='$deletelid'";

        $ret = sqlInsertClean_audit($sql);

}
```

As you can see the ID string is just included directly into the string used for the query.

As a proof of concept, I wrote a simple SQL extract that when injected produces a valid but nefarious query. In this case, the query deletes all Patient Disclosures.

The malicious Request URL might look like this (the malicious characters in red):

http://youropenemrserver/openemr/interface/patient_file/summary/disclosure_full.php?deletelid=5**%27%20OR%20%271%27=%271**

The active code inserted is:

’ OR ‘1’=‘1

This generates a SQL query like this:

delete from extended_log where id=‘5’ OR ‘1’=‘1’

The addition ensures every item in the table is deleted. Not only those with an id of 5. Other injections are of course possible, this one was chosen because its a simple demonstration of SQL Injection. Typically an attacker would try to extract user credentials, or confidential information - in this case possibly patient medical records.

One positive aspect of the flaw is that it is not pre-auth. So the attack only works when the attacker/exploit code has access to a valid logged-in session. This makes it slightly harder to exploit, but not overly so as an attacker can use methods such as Cross Site Request Forgery to initiate ‘blind’ attacks from another browser tab. But in summary, if OpenEMR is deployed only on a local network this issue is not severe.

Note: I reported this issue in a process of responsible disclosure on a 30 day embargo. (That expired 5 days before a patch (http://www.open-emr.org/wiki/index.php/OpenEMR_Patches) was released and 9 days before this post.

The patch was released on the 8th June 2014 and is meant to address this issue and others. (Look for the fixes from Brady Miller to log.inc (http://www.open-emr.org/wiki/index.php/OpenEMR_Patches#List_of_files_.284.1.2.29)). I have not tested this fix.